On Reducing Redundancy and Improving Efficiency of XML Labeling Schemes

Changqing Li, Tok Wang Ling, Jiaheng Lu and Tian Yu
Department of Computer Science, National University of Singapore, Singapore, 117543
{lichangq, lingtw, lujiahen, yutian}@comp.nus.edu.sg

ABSTRACT
The basic relationships to be determined in XML query processing are ancestor-descendant (A-D), parent-child (P-C), sibling and ordering relationships. The containment labeling scheme can determine the A-D, P-C and ordering relationships fast, but it is very expensive in determining the sibling relationship. The prefix labeling scheme can determine all the four basic relationships fast if the XML tree is shallow. However, if the XML tree is deep, the prefix scheme is inefficient since the prefix is long. Furthermore, the prefix_label is repeated by all the siblings (only the self_labels of these siblings are different). Thus in this paper, we propose the P-Containment and P-Prefix schemes which can determine all the four basic relationships faster no matter what the XML structure is; meanwhile P-Prefix can reduce the redundancies in the prefix labeling scheme.

Categories and Subject Descriptors
H.2.4 [**Database Management**]: Systems – *query processing*

General Terms
Performance.

Keywords
XML, Labeling scheme, Query, Redundancy.

1. INTRODUCTION
XPath and XQuery are two main XML query languages. The following XPath query:

*/book[/title]//section/paragraph[2]/preceding-sibling::**

finds all the elements that are siblings of *paragraph[2]* (means the *second paragraph*) and these sibling elements should be before *paragraph[2]*. Meanwhile, *paragraph[2]* should be a child of *section* and *section* should be a descendant of *book*. In addition, *book* should satisfy the restriction that it has a child *title*.

"//" represents the A-D relationship, "/" represents the P-C relationship, and "*preceding-sibling*" represents two relationships i.e. sibling relationship and ordering relationship (preceding).

Therefore to facilitate the XML queries, the core operation is to efficiently determine these four basic relationships. The labeling (numbering) [2, 3] schemes can help to determine these relationships, but each labeling scheme is not efficient to determine all the four basic relationships. For instance, the containment scheme [3] is very inefficient to determine the sibling relationship; it needs to search the parent of a node, then decide whether another node is a child of this parent. The prefix scheme [2] is inefficient in determining all the four relationships if the XML tree is deep. Thus the objective of this paper is to propose labeling schemes that can efficiently determine all the four basic relationships no matter what the XML structure is.

The main contributions of this paper are summarized as follows:

- We propose the P-Containment scheme which can efficiently determine all the four basic relationships.
- We propose the P-Prefix scheme which can reduce the redundancies in the prefix scheme, and determine all the four basic relationships efficiently even if the XML tree is deep.

2. RELATED WORK AND MOTIVATION
We use examples to illustrate the labeling schemes.

2.1 Containment Scheme
Example 2.1. *Figure 1 shows the containment labeling scheme [3]. The values near each node are the "start", "end" and "level" values. "5,6,3" is a child of "2,7,2" since interval [5, 6] is contained in interval [2, 7] and levels 3 – 2 = 1. To determine whether "5,6,3" is a sibling of "3,4,3", the containment scheme needs to search the parent of "3,4,3" firstly, then decide whether "5,6,3" is a child of this parent. The search of the parent needs a lot of parent-child determinations and is very expensive.*

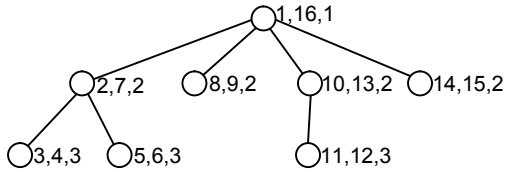

Figure 1. Containment scheme.

2.2 Prefix Scheme
Example 2.2. *It is inefficient for the prefix scheme [2] to determine all the four basic relationships if the XML tree is deep. For instance, to determine that "1.2.1.1.3.3.4.5" is a parent of "1.2.1.1.3.3.4.5.2", the prefix scheme needs to compare 8 pairs of numbers. In addition, the prefix_label should be repeated by all the siblings. For instance, "1.2.1", "1.2.2" and "1.2.3" repeat the prefix_label "1.2" three times. These are redundancies.*

CIKM'05, October 31-November 5, 2005, Bremen, Germany.
ACM 1-59593-140-6/05/0010.

3. P-CONTAINMENT AND P-PREFIX

3.1 P-Containment Scheme

Different from the traditional containment scheme [3], we store the "parent_start" value rather than the "level" value. The "parent_start" value of a node is the "start" value of its parent. We call this improved containment scheme P-Containment. Based on P-Containment, we can determine the parent-child relationship faster, and determine the sibling relationship much faster.

Property 3.1 *For two **different** nodes u and v, node u is a parent of node v iff the "**parent_start**" value of node v is equal to the "**start**" value of node u based on P-Containment.*

Property 3.2 *For two different nodes u and v which are **not** the **root** of the XML tree, node u is a sibling of node v iff the "**parent_start**" value of node u is equal to the "**parent_start**" value of node v based on P-Containment.*

The ancestor-descendant and ordering relationship determinations based on P-Containment are the same as the traditional containment scheme.

3.2 P-Prefix Scheme

Though the idea of our P-Prefix is to some extent similar to the idea of P-Containment, they have differences.

3.2.1 Speed Up Sibling and Ordering Relationship Determinations (P-Prefix-I)

To reduce the redundancy of the prefix scheme, we separate the prefix_labels and self_labels, remove the *duplicated* prefix_labels appeared later, and give each unduplicated prefix_label a unique index number (called P-Prefix-I).

Property 3.3 (Sibling Determination). *Based on P-Prefix-I, node u is a sibling of node v iff $P\text{-}PIndex_I(u)$. = $P\text{-}PIndex_I(v)$, where $P\text{-}PIndex_I$ means the P-Prefix-I index.*

Property 3.4 (Ordering Determination). *Based on P-Prefix-I, node u is before (after) node v in document order iff 1) $P\text{-}PIndex_I(u) < (>$ resp) $P\text{-}PIndex_I(v)$; or 2) $P\text{-}PIndex_I(u) = P\text{-}PIndex_I(v)$ and self_label(u) < (> resp) self_label(v).*

P-Prefix-I guarantees that the sibling relationship determination is only one comparison and the ordering relationship determination is at most two comparisons no matter how deep the XML tree is.

3.2.2 Speed Up Parent-Child Relationship Determination (P-Prefix-II)

The main idea to determine the P-C relationship is that we store the parent index of a node together with the index of this node (similar to P-Containment), called P-Prefix-II. If the parent index is built on the labels instead of the prefix_labels, the parent-child relationship determination only needs one comparison, i.e. the parent index of one node is equal to the index of another node. But in that way, the sibling and ordering relationship determinations are expensive when the XML tree is deep. Thus *based on P-Prefix-I*, we build the parent index on *prefix_labels*.

Definition 3.1 (Second_self_label). *A label is a second_self_label if it is the self_label of a prefix_label.*

Definition 3.2 (Second_prefix_label). *A label is a second_prefix_label if it is the preifx_label of a prefix_label.*

Property 3.5 (P-C Determination). *Node u is a parent of node v iff $P\text{-}PIndex_I(u) = P\text{-}PParentIndex_I(v)$ and self_label(u) = second_self_label(v), where $P\text{-}PParentIndex_I$ means the parent P-Prefix-I index.*

Property 3.5 guarantees that the P-C determination is only two comparisons no matter how deep the XML tree is and the sibling and ordering determinations are still the same as P-Prefix-I.

3.2.3 Speed Up Ancestor-Descendant Relationship Determination (P-Prefix-III)

To facilitate the ancestor-descendant relationship determination, based on P-Prefix-II, we index the second_prefix_label for every certain number depth, called P-Prefix-III index. Based on P-Prefix-III index, we can determine the A-D relationship at a higher level firstly, then at a lower level.

Definition 3.3 (Remainder_second_prefix_label). *Suppose the total depth of the second_prefix_label is TD and we index every DI depth, then the remainder_second_prefix_label is the rest TD mod DI depth of the second_prefix_label.*

Property 3.6 (A-D Determination). *(This property is a procedure) Suppose the label is in sequence $P\text{-}PIndices_{III}$ $\oplus$ remainder_second_prefix_label $\oplus$ seond_self_label $\oplus$ self_label, where $P\text{-}PIndices_{III}$ are the P-Prefix-III indices. We directly compare the labels of nodes u and v from left to right. If the comparison is between an P-Prefix-III index and a label, we get back the label based on the P-Prefix-III index and continue the comparisons. If label(u) is a prefix of label(v), node u is an ancestor of node v.*

P-Prefix-III can determine all the four basic relationships faster.

4. PERFORMANCE STUDY

We test P-Containment and P-Prefix. P-Containment works faster than the traditional containment scheme to determine the parent-child and sibling relationships. P-Prefix can determine all the four basic relationships faster even if the XML tree is deep; meanwhile P-Prefix has smaller label size than the traditional prefix scheme.

5. CONCLUSION

In this paper, we have proposed the P-Containment and P-Prefix schemes which can determine all the four basic relationships very fast no matter what the XML structure is. In addition, P-Prefix reduces the redundancies of the traditional prefix scheme. More details of this paper can be found in [1].

6. REFERENCES

[1] C. Li and T.W. Ling. On Reducing Redundancy and Improving Efficiency of Labeling Schemes (long version of this paper). *Available from the author*, 2005.

[2] I. Tatarinov, S. Viglas, K.S. Beyer, J. Shanmugasundaram, E.J. Shekita, and C. Zhang. Storing and querying ordered XML using a relational database system. In *Proc. of SIGMOD*, pages 204-215, 2002.

[3] C. Zhang, J.F. Naughton, D.J. DeWitt, Q. Luo, and G. Lohman. On Supporting Containment Queries in Relational Database Management Systems. In *Proc. of SIGMOD*, pages 425-436, 2001.